Compact and Progressive Plant Models for Streaming in Networked Virtual Environments

Sébastien Mondet, Wei Cheng, Géraldine Morin, Romulus Grigoras, Frédéric Boudon, Wei Tsang Ooi

Compact and Progressive Plant Models for Streaming in Networked Virtual Environments

SEBASTIEN MONDET
University of Toulouse, France
and
WEI CHENG
National University of Singapore, Singapore
and
GERALDINE MORIN and ROMULUS GRIGORAS
University of Toulouse, France
and
FREDERIC BOUDON
CIRAD, Montpellier, France
and
WEI TSANG OOI
National University of Singapore, Singapore

Just as in the real world, plants are important objects in virtual worlds for creating pleasant and realistic environments, especially those involving natural scenes. As such, much effort has been made in realistic modeling of plants. As the trend moves towards networked and distributed virtual environments, however, the current models are inadequate as they are not designed for progressive transmissions. In this paper, we fill in this gap by proposing a progressive representation for plants based on generalized cylinders. We model the shape and thickness of branches in a plant as Bézier curves, group the curves according to the similarity, and differentially code the curves to represent the plant in a compact and progressive manner. To facilitate the transmission of the plants, we quantify the visual contribution of each branch and use this weight in packet scheduling. We show the efficiency of our representations and the effectiveness of our packet scheduler through experiments over a wide area network.

Categories and Subject Descriptors: I.3.2a [**Graphics Systems**]: Distributed/Network Graphics; C.2.4b [**Distributed Systems**]: Distributed Applications

General Terms: Design, Performance, Experimentation

Additional Key Words and Phrases: Streaming, Plant models, Multiresolution, Progressive coding, Progressive transmission, Networked Virtual Environment

1. INTRODUCTION

Networked virtual environments (NVEs) have become an increasing popular class of applications, due to advances in graphics capability of commodity desktop. Successful examples include Second Life and Google Earth. Such applications typically describe a 3D virtual world using a collection of media data (3D models, animation, images, audio, and video) stored on a remote server. A user connects to the server and requests for a subset of the media data depending on its current viewpoint. Upon receiving the requested data from the server, the client creates and renders a partial/local 3D scene, which the user can then interact with and navigate through.

Plants are important and common objects in a networked virtual environment. Just as in the real world, plants help create a pleasant and realistic virtual world, especially those involving natural scene. Billboards and L-systems are common methods to represent plants within virtual worlds. While these simple representations suffice for NVE applications such as games, realistic and botanically-accurate representations of plants are crucial in NVE applications such as virtual forests or virtual botanical gardens, where users are expected to inspect a plant closely and possibly interact with plants. In certain scenarios, it is not enough to model a particular species of plants. Rather, an individual famous or historical plant needs to be modeled. Two examples of such plants are the Bodhi Tree in Bodh Gaya, India, or the "five-dollar" Tembusu tree in Singapore Botanical Garden. Image-based representations and randomly generated plants cannot meet the level of details and botanical accuracy demanded by these applications.

Previous work has focused on how to accurately model a plant [Remolar et al. 2002; Bloomenthal 1985; Prusinkiewicz et al. 2001; Prusinkiewicz and Lindenmayer 1990; Neubert et al. 2007] or easily create a plant[1] for a virtual environment. Realistic and detailed plant models can require up to hundreds of thousands of polygons. Remolar et al. [Remolar et al. 2002] estimated that a plant generated by XFrog, a well known plant modeling platform, can consist of 50,000 polygons to represent the branches. The plants can have 20,000 or more leaves, which themselves consist of polygons. Neubert et al. [Neubert et al. 2007] reported the plant models that they used consist of up to 555,000 polygons. These numbers are for a single plant. In natural scenes, such as forests, one would expect the scene to contain tens to hundreds of plants. As we scale a virtual environment to large number of detailed plants, the download time increases and may exceed the tolerable response time needed in an interactive environment. Waiting for completed download of plants before rendering them on screen becomes infeasible.

We can reduce the waiting time by compromising on the rendering quality of plants using a progressive representation of plants. With a progressive representation, a plant can be stream progressively – a coarse version of the plant is sent first, followed by a series of refinements to incrementally improve the level of details of the plant. The receivers can thus render a less detailed version of the plant, and progressively add further details as the refinements are received.

Progressive representation for general 3D objects, such as progressive meshes [Hoppe 1996] are well studied. These representations, however, are not suitable

[1]Dryad (http://dryad.stanford.edu)

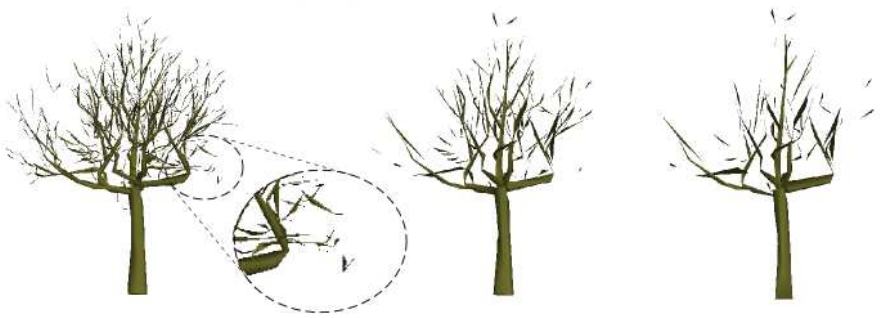

Fig. 1. Mesh simplification on a *Walnut* model. The original model consists of 278,632 triangles. From left to right: models consisting of 1%, 0.2% and 0.1% of the original model.

for plants due to the topology structure of the branches. Removing triangles to simplify the plants without affecting the topology becomes difficult beyond a certain level. Representation of plants by progressive meshes thus gives unsatisfactory results [Remolar et al. 2002]. Figure 1 illustrates that the simplification of a tree represented with progressive mesh does not preserve the topology, in particular, the connectivity of the tree. Hence, progressive representations that suit the topology of plants are needed.

In this paper, we propose a progressive representation of plants that *preserves* the branching structure of a plant, even at a very low resolution. We focus on representing branches of a plant, and do not explicitly consider leaves in this paper. Our proposal is based on a skeletal representation of the tree, organized hierarchically into a data structure called n-tree and rendered as a set of generalized cylinders. Further, our representation can be differentially coded for efficiency.

The branching structure and differential coding introduce dependencies among the data representing a plant. These dependencies need to be accounted for when streaming our plant representation in a lossy environment. If a packet is lost: a branch depending on this packet cannot be decoded, even if it has already been received. The decoding of the branch will have to wait for the retransmission and reception of the lost packet. This situation causes delay in rendering and should be avoided especially in interactive applications. Furthermore, each branch contributes a different amount to the visual quality of the rendered plant. Therefore, information for branches that are more visually prevalent should be sent with higher priority.

In this paper, we adopted our previously proposed framework for progressive mesh streaming [Cheng, Ooi et al. 2007] to streaming of plants. Our original framework can be generalized in order to stream any partially ordered data, in particular, our proposed plant representation. The framework considers the characteristics of the 3D model (dependencies and visual contributions) as well as the characteristics of the network (retransmission delay, packet loss rate), and allows us to estimate the quality of the rendered 3D model at a given time. From this information, a packetization strategy and a sending order is determined to maximize the quality of the decodable model at the client. To adopt our previous framework to our new representation of plants, we need to quantify the importance of

each component of the progressive representation. In this paper, we measure the importance of a branch by its size and its distance from the viewpoint.

We evaluate our method using a complex digitized walnut tree [Sinoquet et al. 1997] (referred to as *Walnut* in the rest of this paper) and an apple tree [Costes et al. 2003] containing irregularities of a real, natural, branching topology, as well as using more regular trees generated using L-systems [Prusinkiewicz and Lindenmayer 1990]. We show that our representation can be compressed efficiently, retains progressivity, and achieves good rendering quality during transmissions.

Our major contributions in this paper are (i) an efficient and effective new progressive representation for branching structures in a plant, and (ii) an effective method for streaming this new representation over lossy network for applications such as networked virtual environments.

The rest of this paper is structured as follows. Section 2 reviews the state-of-the-art in 3D model streaming and plant representation. Section 3 describes our proposed representation. We evaluate the suitability of our proposed scheme in Section 4, and finally, we conclude in Section 5.

2. STATE OF THE ART

Before we present our work on progressive streaming of plants, we first review the existing literature on progressive streaming of 3D models and representation of plants.

2.1 Streaming of 3D Models

Previous research in streaming of 3D models has considered many different 3D representations and different aspects of the problem. A common theme of these work is how to improve the quality of the rendered mesh, given that the network is lossy.

One way to improve the quality is to encode the 3D models in a way that is resilient to losses. Park et al. and Yan et al. propose error resilient compression of progressive meshes to accomplish this goal [Park et al. 2006; Yan et al. 2001] through appropriate segmentation of the mesh to prevent error propagation. Point based representation of an object is inherently loss resilient, and is used by Rusinkiewicz and Levoy [Rusinkiewicz and Levoy 2001] and Tarin et al. [Tari et al. 2005]. Others have considered error control mechanisms for improving the rendered mesh quality [Al-Regib et al. 2002; Chen et al. 2005].

Using the right transport protocols can appropriately trade off between delay and robustness. This issue is considered by Li et al. [Li et al. 2006], Al-Regib and Altunbasak [Al-Regib and Altunbasak 2003], Harris III and Karvets [Harris and Kravets 2002].

Packet scheduling can affect the rendered quality of the 3D models as well. Ramanathan et al. [Ramanathan et al. 2003] extend rate-distortion framework to streaming of light fields. Cheng et al. [Cheng, Ooi et al. 2007] propose a greedy heuristic in deciding which vertex splits to send first when streaming a progressive mesh. Yang et al. [Yang et al. 2004] allocates bandwidth between mesh data and textures appropriately to improve the rendered quality.

One can send only segments of the 3D models that are visible to the viewers. Such view-dependent streaming has been explicitly or implicitly considered in these

previous work. For instance, Meng and Zha [Meng and Zha 2003] use user's gaze to guide the transmissions of point-based models. Cheng and Ooi [Cheng and Ooi 2008] consider how to estimate the visibility and visual contributions of vertex splits at the receiver.

An important issue to consider is how to model the quality of the 3D model. Quantifying the visual contribution of a data unit can help deciding which data unit to send. Cheng and Ooi [Cheng and Ooi 2008] estimate the quality of a vertex split in a progressive mesh using the screen-space area of the faces around the vertex split. In [Ramanathan et al. 2003], authors measure the distortions of images in light fields. Tian and Al-Regib [Tian and AlRegib 2004] and Cheng et al. [Cheng, Ying et al. 2007] propose metrics to quantify the quality of a simplified mesh with textures.

However, as presented above, the literature on streaming of 3D models mainly concentrate on mesh-based, point-based, or image-based representation of *generic* 3D objects. While each of these representations can be optimized to represent different types of objects within a virtual environment, none of them can effectively and progressively represent realistic plants. In the next section, we briefly introduce the current state-of-the-art in plants representation.

2.2 Representation of Plants

Plant geometry is particularly complex and thus motivated a variety of representations dedicated to its specific needs [Deussen and Lintermann 2005; Boudon et al. 2006]. Branches and foliage are usually treated separately. From a modeling point of view, a string representation of the branching structure is coupled with rewriting rules, called L-systems [Prusinkiewicz and Lindenmayer 1990], to simulate the growth of the plant, and with a LOGO style turtle that interprets the symbols of the string as geometric commands [Prusinkiewicz 1986]. In this system, the geometry of a symbol is built according to the geometry of previous elements. In this case, leaves are instances at different places of the same geometric symbol. This idea inspired the representation we use in this work. More generally, some high level representations for branches have been proposed based on parametric [Bloomenthal 1985] or implicit surfaces [Galbraith et al. 2004]. They rely on a branching skeleton which is extended with radius (given by cross sections or implicit functions). Skeleton is defined as a set of connected parametric curves. These branching structure representations have the advantage to be compact compared to more discrete representations such as mesh and provide support for animation (which is not the case of the simplified models whose connectivity is lost in Figure 1). By default, however, they are not adapted for progressive description. The goal of this paper is precisely to fill this gap.

From a rendering point of view, some representations based on images [Meyer et al. 2001; Decaudin and Neyret 2004; Behrendt et al. 2005], points [Weber and Penn 1995; Deussen et al. 2002] or polygons [Remolar et al. 2002; Zhang et al. 2006] proposed adaptive schemes for displaying trees. These representations mainly focus on foliage (leaves) and thus can be seen as complementary to ours since they are usually complemented with polygonal representations of trunk and branches. The representations that offer some interesting results, however, usually require a large amount of data, in particular those with points and images. Polygonal rep-

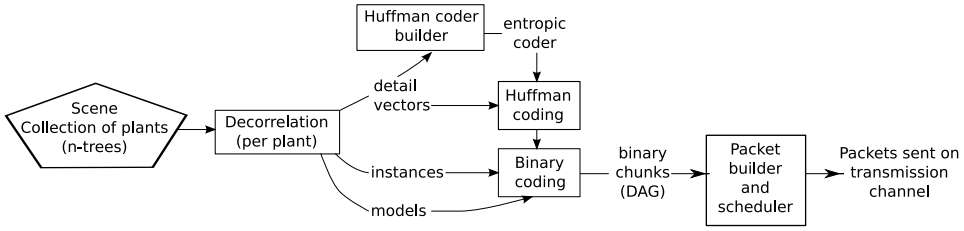

Fig. 2. The encoding process for a model based on skeletal representation.

resentations on sparse geometry such as foliage are not totally convincing. These representations can be streamed with classic methods since they use classic primitives with low-level abstraction. By default, however, they seem more dedicated to static representations. They have to be attached to a skeleton representation to support animation.

3. STREAMABLE REPRESENTATION OF PLANTS

The lack of suitable, progressive, and dynamic representation of plants that allow plants to be streamed and rendered at multiple level of details motivates our work in this paper. In this section, we present our proposed representation of plants. We then discuss how to code, compress, and stream it.

Figure 2 outlines the steps from encoding to streaming of our representation and guides the presentation of this section. Our starting point is a natural scene using a plant model based on skeletal representation (Section 3.1). This representation serves as the basis for our proposed compressed, progressive representation that decorrelates information into three components called *models*, *instances*, and *detail vectors*. The detail vectors are compressed with entropy coding (Section 3.2). We then convert the plant model into binary chunks. Each chunk is assigned an *importance* value, which is then used in packetizing and scheduling the chunks for streaming (Section 3.3).

3.1 Initial Plant Model

Our representation focuses on the branching structure of a plant and is thus based on a skeletal representation. A branch is represented by (i) an axis curve, modeled as a Bézier curve of degree d, and (ii) a radius along the branch, modeled as a $2D$ Bézier curve. Such generic high level representation can then be displayed as generalized cylinders [Bloomenthal 1985; Prusinkiewicz et al. 2001] (which is the case in this paper) or implicit surface [Galbraith et al. 2004]. This skeletal representation is much more compact than a mesh representation. For example, the *Walnut* at full resolution only requires 10772 control points using our representation compared to 278,632 triangles using a mesh model. The chosen generalized cylinders warrant the topology of the tree, that is, the connectivity of the branches. Thus, this model may be used for animation: kinetic information could be added on the skeleton structure (not addressed in this paper).

The branches are organized inside an n-ary tree data structure giving the structure of the plant. We call such a data structure an n-tree, to avoid confusion with the concrete plant object that we model. The root of the n-tree is the trunk of the

plant and branches borne by the trunk are the n-tree children of this trunk. Each child branch contains a parameter u ($0 \leq u \leq 1$) giving the position of the attachment point on its bearing parent branch [Prusinkiewicz et al. 2001]. The parameter u defines the first control point of the Bézier curve of the child branch. The d remaining control points are encoded in the child branch by their three coordinates in space (c.f. Figure 3).

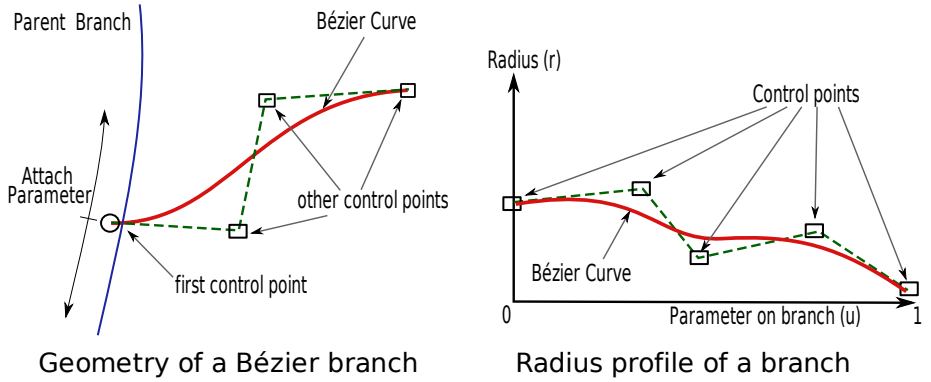

Fig. 3. On the left, the Bézier curve representing a branch with its attachment parameter (u) on its parent branch. On the right, the Bézier curve representing the radius along a branch (i.e. $(u, r) \in [0, 1] \times [0, \infty]$).

An arbitrary attribute of the branch may be defined as a function of the parameter u on the interval $[0, 1]$. Here, the radius of the branch illustrates how attributes along the branch are coded. It is defined as a positive real value along the branch. To model it as a smooth function along the branch, we represent its values as a series of control points (u_i, r_i) of a Bézier curve of degree m, where $(u_i)_{i=0...m}$ is an increasing sequence in the interval $[0, 1]$ that defines the location of the branch, and $(r_i)_{i=0...m}$ characterizes the radius for the corresponding given location. Note that the degree m of the radius curve is not related to the degree d of the bearing branch.

3.2 Compact Progressive Models for Plants

3.2.1 *Multi-resolution model.* To encode a plant as a compressed multi-resolution representation, we exploit the similarity between different branches and between their radii. We can view the compression of branches and radii as compression of two sets of Bézier curves. The idea of the compression algorithm is to first group similar Bézier curves together, compute an *average* curve for each group, and encode the differences between the control points of a Bézier curve and the average curve. Since we group similar curves together, these differences are small and may be quantized with fewer bits, leading to a compact coding.

A key question is thus how to best group the Bézier curves such that the differences within a group is small. We discuss this issue in details in the Section 3.2.2.

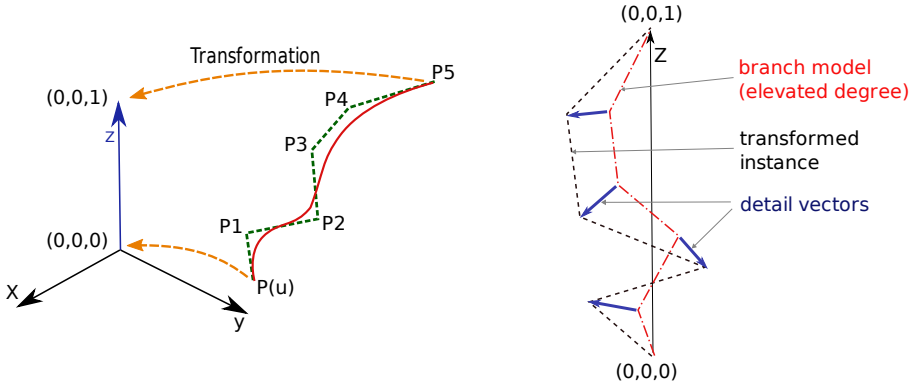

Fig. 4. On the left, the standardization of a Bézier curve. On the right, a standard representation of a branch (regular dash), its model -after degree elevation for the case of curves of degree 2- (red irregular dash), and the detail vectors.

The clustering process gives us clusters of Bézier curves. We can then create multi-resolution representation of the curves from each of the clusters. In the following, we elaborate on this process.

Standardization. To compare and code the differences between two Bézier curves, we need a *standard representation* of these curves. For branches, we use an affine transformation to convert back and forth between an original branch and its standard representation. The affine transformation is defined so that P_0 and P_N, the first and last control points of the original Bézier curve, map to the origin $(0,0,0)$ and the point $(0,0,1)$ respectively (c.f. Figure 4). We characterize this first mapping by two rotation angles and a uniform scaling factor. Since we choose to apply a uniform scaling, there is a degree of freedom remaining, which corresponds to the rotation around the z, to completely define the affine transformation. We fix the rotation around the z axis so that the center of gravity (or average) of the control points, P^b, lies in the xz plane.

More specifically, the affine transformation that maps a branch shape to its standard representation is characterized by

—a translation of vector $t = -\overrightarrow{P_0}$,

—a scaling factor $s = \frac{1}{\left\|\overrightarrow{P_0 P_N}\right\|}$,

—three rotation angles such that

$$\overrightarrow{T(P_0)T(P_N)} = \vec{z} \text{ and } \overrightarrow{T(P_0)T(P^b)} \cdot \vec{y} = 0,$$

where $T(P)$ denotes the image of P by the affine transformation.

For radii, since the parameters u_i already fall in the normalized interval $[0,1]$, we normalize the family (r_i) by dividing it by the average norm of the radii $\frac{\sum_i r_i}{m+1}$.

In the following, the standard representation will refer to either the control points of a normalized branch, or, to the control points of radius with normalized r_i.

Models. We can now calculate the average Bézier curve using the standard representation. The average Bézier curve of a set of standardized Bézier curves with a common degree d is a Bézier curve of degree d, such that its i-th control point $\overline{P_i}$ is the barycenter of the i-th control points of the standardized curves, mapped from the curves of degree d. We call this average Bézier curve the *branch model*, for branches, and the *radius model*, for radii.

Instances. Each branch can be approximately represented as soon as its instance is known. The instance of a branch consists of the attachment parameter u, the transformation to the standard representation, the radius scaling factors along the branches, as well pointers to the bearing branch and the models (see Section 3.2.3).

Detail Vectors. We now code each Bézier curve in *differential form* relative to the model, storing its differences to the corresponding control point of the model ($\overline{P_i}$) (c.f. Figure 4). We call the differences *detail vectors*. It is thus possible to encode these detail vectors using a limited number of bits. It should be noted that since the curves are normalized, the first and last control points do not need to be coded. For example, Bézier curves representing branches of degree 3 only need two intermediate points to be defined. The encoding of a branch, and similarly, of a radius, is now defined by a set of instantiation parameters (transformation to standard representation) and a set of differential data (from the model).

Our representation allows branches and radii of a plant to be displayed progressively in two ways. First, parent branches are displayed before their children branches and descendants. Second, the branch instances are displayed first with the corresponding instance radius, showing an approximate shape of the branches. The detail vectors of a branch and its radii may refine its shape afterward.

3.2.2 *Grouping Policy.* We group the Bézier curves so that we can differentially code the curves as detail vectors instead of the original control points. How we group the curves affects not only the compression ratio, but also the intermediate visual quality of the tree and the quantization error. We have implemented several grouping strategies, each to satisfy these different criteria: compression, quantization, or the visual aspect. Note that these criteria apply for both the original (full resolution) model and the intermediate, partially rendered, models. The different strategies may be considered as cascading filters, as shown in Figure 5.

A first simple (successful) strategy [Mondet et al. 2008] groups branches according to the degree d of their curves to simplify curve comparison. In that case, the branch model, as well as the details, have $d-1$ control points. In order to minimize the quantization error, we propose to add a hierarchical clustering algorithm [Johnson 1967]. Clustering is applied on the Bézier curves by first defining an initial distance between every two curves. Then, a greedy procedure merges the clusters two by two, choosing, at each step, the smallest distance until the desired number of clusters is reached. At each merge, the distances to the newly created cluster are easily computed using a *link function*, which computes the distance to the new cluster from the distances to the two original clusters.

A second strategy is to remove the constraint on grouping according to the degree d, by applying a degree reduction algorithm [Bogacki et al. 1995]. In practice, any Bézier curve of degree $m \geq 2$ is approximated by a curve of degree 2. From the deterministic degree raising algorithm (see for example [Farin 2002]), an ap-

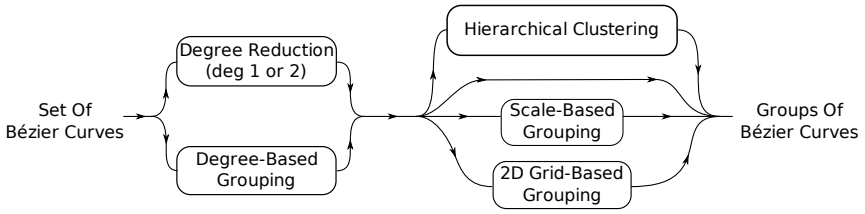

Fig. 5. The grouping methods seen as cascading filters.

proximating curve of degree m is computed. The detail vectors are the difference to this approximating curve (as in Figure 4). This step corresponds to the filter *Degree Reduction* in Figure 5. Curves of degree 1 are treated separately and have their own model. The degree reduction algorithm we use ensures that end points do not move. This is an important condition for us, since the end points for the branches' curves are implicitly $(0, 0, 0)$ and $(0, 0, 1)$ and therefore are not coded. The degree reduction has two advantages. First, by grouping Bézier curves into only two groups (degree 1 and 2), further clustering may be applied since curves are partitioned in bigger groups. Second, models now only have zero or one control point to code for the branches, and two or three control points to code for the radii; they are therefore much lighter.

An additional grouping strategy, called *scale-based*, uses the length of the branch (or equivalently, the scaling factor s of the standard representation) to create the groups. Let s_{max} be the scaling factor of the longest branch and g be the number of groups that we wish to create. We partition the scaling factors uniformly and created groups with scaling factors $(0, s_{max}/g], (s_{max}/g, 2s_{max}/g], \ldots ((g-1)s_{max}/g, s_{max}]$. As a typical tree has fewer long branches and more short branches, longer branches tends to be grouped with few branches, while shorter branches are grouped into bigger groups. Since short branches are likely not to bear children branches, having a less accurate version of these branches in the intermediate tree is visually acceptable. Otherwise, the shape of a bearing branch would affect all children branches, causing popping effects. Such scale-based grouping not only preserves to good compression, but gives better visual results for the progressiveness of the tree (see Section 4.1).

Finally, another grouping strategy using the geometric position of the middle point in the approximating degree 2 Bézier curve has been tested. The result are not as convincing as the scale-based partitioning.

In Section 4.1, we present some experiments on the grouping policies. These experiments lead us to choose a *best compromise* grouping strategy, which consists of reducing the degree of Bézier curves, and grouping the branches' shapes by scale (with $g = 4$). Obviously our choice can be challenged: other performance criteria or other experimental data (plant models) may produce a different best compromise. Nevertheless, we are using this "*best compromise*" setup for the rest of the experiments.

3.2.3 *Dependencies in the progressive representation.* In order to efficiently handle a large model (e.g. load it into memory or transmit it over the network) with

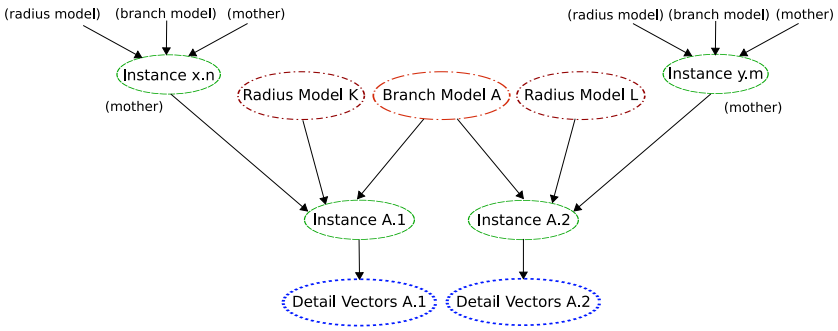

Fig. 6. One dependency level in the progressive representation of a plant.

a progressive representation of the branch system, we need to express the dependencies between pieces of data: A depends on B meaning that the decodability of A *requires* that B has already been decoded.

There are two main families of dependencies: topological dependencies and those generated by the differential coding. The first family is related to the n-tree structure of the plant: a branch depends on the parent branch it attaches to. A radius depends on its branch. The second family includes the dependencies due to differential coding, that is, the dependence between an instance and its two models (branch and radius), and between a set of detail vectors and its corresponding instance.

Figure 6 shows these three types of dependencies. Instance $A.1$ depends on its parent branch instance $x.n$ (topological dependence). Note that a child branch is independent of the detail vectors of its parent branch; Section 3.3.2 uses this independence and shows how we prioritize between child branches and detail vectors. Instance $A.1$ also depends on its branch and radius models (A and K). The set of detail vectors $A.1$ depends on the instance $A.1$. These two last kinds of dependencies appear with differential coding.

The progressive representation created by the model, instance, and detail vector chunk and their dependencies does not contain cycles and therefore can be modeled as a *Direct Acyclic Graph* (DAG).

3.2.4 *Quantization of Details.* One advantage of multi-resolution differential coding is the ability to (i) quantize small detail vectors with a small number of bits and (ii) to choose accurate binary representative *symbols* according to their distribution. In this section, we first show that the evaluation of our resulting detail vectors leads to a beneficial usage of an entropy coder, and then, we present our quantization method followed by our implementation of a Huffman coder for encoding the detail vectors.

To evaluate the accuracy of using an entropy coder in our method, we have computed, for a given quantization (i.e. a given number of bits per floating point number), the induced error and the theoretical entropy of the represented data. The maximal induced error gives the accuracy of the quantization, while the computed theoretical entropy gives the mean number of bits to expect after Huffman coding.

The quantization can be vector or scalar. We have carried out experiments with both methods. We have started with vector quantization [Mondet et al. 2008].

The vector quantization is carried out in two steps. First we compute the AABB (Axis-Aligned Bounding Box) of all detail vectors (by finding the min and max of the x,y,z coordinates). Then, to quantize each coordinate into c bits, we build a 3D grid corresponding to 2^{3c} vectors uniformly distributed in the AABB. Each detail vector is then represented by the symbol of the nearest of the vectors discretized on the grid. The quantization error is thus the distance from the quantized vector to the original detail vector. To reconstruct the quantized vectors, a header containing the AABB of the vectors (6 floating point numbers) and the number of bits per coordinate is sufficient.

The resulting error for a given number of bits per coordinate could still be decreased by processing a few iterations of a classification algorithm such as k-means. However, the resulted gain would be offset by increased header size, since transmission of the actual values of the representing symbols chosen by the classification would be necessary.

After analyzing vector quantization performance, we have noticed that

—the weight of the dictionary is important using vector quantization on our data; and

—difference vectors do not show a privileged direction (or a high density) and the values of their scalar components are quite close.

Therefore we have made experiments with classic scalar quantization (i.e., vector quantization in dimension 1) over the set of all scalars representing the components of detail vectors (for branches and radii). The results show that scalar quantization performs better. Even though vector quantization is slightly better at reducing the entropy, the gain does not compensate the higher header overhead. Additionally, even if 3D-vector quantization was used for branches, we still would need to use 2D or scalar quantization for radii. The results presented in this paper only use scalar quantization for details vectors for both branches and radii.

Once each detail vector coordinate is mapped to a symbol we can build the entropy coder. First, we build an entropy histogram, giving the number of represented scalars per symbol (i.e. the probability of each symbol). We have summarized the resulting plots for one sample tree (the *Walnut*, c.f. Section 4) in Figure 7. To improve plot readability, we sort the symbols in increasing order of probability, and we only show effectively used symbols. The shape of each curve shows very promising entropic coding capabilities – a few symbols represent most of the detail vectors.

From the built histograms we can compute the theoretical (Shannon) entropy with the formula:

$$H = -\sum_{w_i>0} w_i \log_2 w_i$$

where w_i is the weight of the i-th represented value (i.e. its probability). The results obtained for *Walnut* are displayed in Table I and in Figure 8. Table I also shows the effective entropy after Huffman coding, which includes the header overhead (Huffman table and parameters).

Values $c = 4$ or $c = 5$ seem to be a reasonable trade-off, but in a *cautious and conservative fashion*, we use $c = 6$ for the rest of the experiments. Lower values,

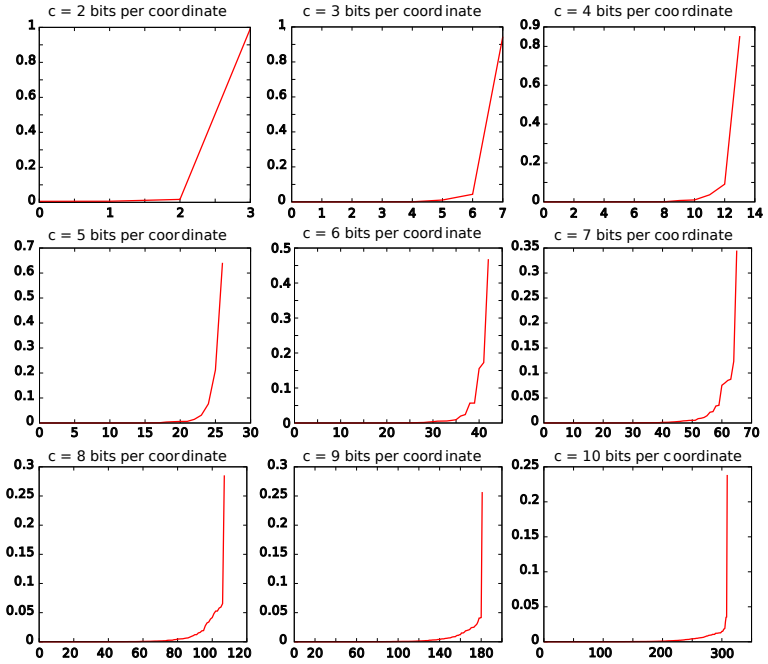

Fig. 7. Entropy histograms computed for $c \in [2, 10]$ show the sorted probabilities of represented values, for the symbols actually used. The repartition of the number of occurrences, corresponding to the shape of the curve, presumes the potential of the entropy coding.

Bits per coordinate	Theoretical entropy	Effective entropy (w/ header)	Error max	Error mean
2	0.094	1.012 (1.017)	0.211	0.059
3	0.429	1.086 (1.092)	0.105	0.051
4	0.768	1.233 (1.240)	0.053	0.017
5	1.655	1.731 (1.742)	0.026	0.016
6	2.455	2.469 (2.488)	0.013	0.008
7	3.243	3.292 (3.324)	0.007	0.003
8	4.037	4.082 (4.137)	0.003	0.001
9	4.816	4.850 (4.948)	0.002	0.001
10	5.695	5.715 (5.890)	0.001	0.000

Table I. Computed entropies and quantification errors, for the *Walnut*, $c \in [2, 10]$ using our *best compromise* setup.

i.e. more aggressive quatization, lead to better compression results, and no *visible* difference. We evaluate the compression efficiency of our method in Section 4.

3.3 Streaming of Plants

The previous section showed how we can progressively represent and code a plant. This section describes how the plant can be streamed. We present how we encode

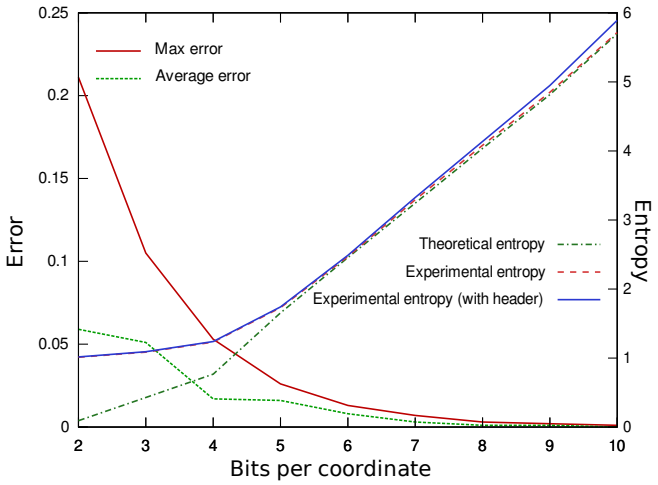

Fig. 8. The entropies and errors for the quantization of the *Walnut* using our *best compromise* setup.

the data into binary chunks and pack the chunks into packets for transmission, a quality metric needed for scheduling as well as a few scheduling strategies.

3.3.1 *Binary Coding.* A progressive representation of a plant consists of four types of data: base data, branch and radius models, instantiation parameters, and detail vectors. Base data contains general characteristics of the plant, information about the trunk, and the entropy coder (the dictionary of symbols). Base data needs to be received first in order to setup the data structures for a plant. Special care has been taken in order to minimize the number of bits used for representing various pieces of information (in particular IDs and pointers).

In order to appreciate the relative weight of various components, details of the *Walnut* model are shown in Table II.

Type	**Number of chunks**	**Size (bits)**			
		min	avg	max	total
Models	7	12	112.57	204	788
Instances	1870	137	137.00	137	256190
Differences	1870	27	50.61	251	94632

Table II. Binary coding: data chunks and their size for *Walnut* coded using our "best compromise" options (Header size: 1150 bits).

Once we encode a plant into binary chunks, the next step is to pack the chunks into packets for transmission. As with packetizing audio and video data, this process packs binary chunks one-by-one into a packet, until the MTU of the packet is reached. The packet is then passed on to the transport layer for transmission. A question that arises here is in what order should the binary chunks be sent. While it is clear that the base data should be sent first, determining the order used for

packing the other type of chunks such that the best rendered quality is achieved at the receiver, is non-trivial. We describe our approach in the next two sections.

3.3.2 *Quality Metric.* First, let us consider the case where there is no packet loss. In this ideal case, the best order for sending the data is to follow the decreasing visual contribution of a chunk – i.e. how much a chunk contributes to the rendered quality of the plant. Doing so would ensure that the receiver can view, at any given time, the plants with the best possible quality.

The question thus is how to quantify the visual contribution, or importance, of a chunk. We describe a quality metric for each chunk as follows:

—the importance of a branch model is a constant k_0,
—the importance of an instance is the value of the scaling factor, corresponding to the size of the branch,
—the importance of detail vectors is the importance of the corresponding instance multiplied by the average length of the detail vectors (detail vectors include coefficients for both the shape of the branch and the radii).

The next question is how to relate these three metrics to each other: we choose to have the importance of instances and detail vectors comparable using two constants (knobs), k_1 and k_2, respectively. For instance, considering that the priority is to increase the density of the branches of the rendered tree, leads to choosing $k_1 \gg k_2$. On the other hand, considering that the shape of the rendered branches is more important than their density during a progressive rendering, leads to choosing $k_1 \ll k_2$.

Figure 9 illustrates the use of these knobs, with those two extreme cases. Intuitively, these parameters can be chosen depending on the application: for a botanist, detail vectors are important for the plant to look realistic; for a computer game player, density of the branches may be of higher relevance.

The figure only shows the static visual influence of the coefficients. One should also note that when detail vectors are delayed too much, a *move popping effect* can be observed as branches which carry many others are deformed when their details are decoded.

The proposed metric is for a single plant. In a scene containing multiple plants, we can adjust the importance of a plant according to its distance from the viewpoint. This importance leads to a simple view-point dependent streaming: plants closer to the viewpoint are streamed first.

3.3.3 *An Analytical Model for Streaming.* For scheduling, two simple strategies may be used: *Naive* which features dependence-only ordering (we send only ready-to-decode data); and *FIFO* which adds importance ordering between binary chunks. *FIFO* can be seen as *almost* optimal in the case of a stream transmission (no packet reordering due to losses). In this section, a more elaborated *Greedy* streaming strategy is presented; it modifies the *FIFO* ordering to take packet loss into account.

When there are packet losses, one needs to consider dependencies in deciding the sending order. Suppose there are two chunks P and Q, with P depends on Q. If we send P and Q separately in different packets, if the packet that contains Q is lost, then P cannot be decoded, even if it is received, until Q is retransmitted. Thus, ideally one should put P and Q into the same packet.

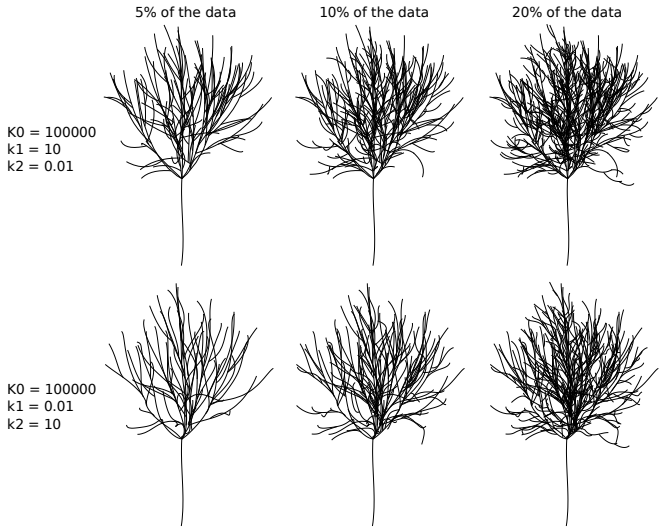

Fig. 9. The influence of the choice of (k_0, k_1, k_2) on the structure of *Walnut* after decoding 5%, 10% and 20% of the data.

The discussion above shows that the ideal order to send the chunks (with the goal of optimizing the quality of the plant), needs to consider both the dependencies and importance of the chunks. The ideal order also depends on network characteristics – packet loss rate, round trip time, and available bandwidth. The latter two parameters determine time to retransmit a loss packet. In our previous work [Cheng, Ooi et al. 2007], we have developed a model for estimating the expected quality of the received 3D model, in the context of progressive mesh. The model, however, is general and can be used for any partially ordered data. In this work, we adopt the model for streaming of plants, by replacing vertices with chunks in the model. We briefly highlight the results from this previous work in the rest of this section for completeness. Interested readers are referred to the original paper for details [Cheng, Ooi et al. 2007].

Our analytical model considers a sender sending packets at an average (normalised) rate of one packet per unit time. We consider retransmission-based protocol. A retransmitted packet always takes precedence over new packets. Let T_d be the average time between sending a packet and discovering that it is lost (either NACK or timeout-based methods can be used). We pack the data to send into packets, and indexed the packets as 1, 2, 3, etc. We let S_i be the time a packet i is sent, and R_i be the time a packet i is received. The average loss rate of the network is p. We can estimate the sending time, receiving time of a packet using the lemmas below.

LEMMA 1. *If* $i \geq T_d$,

$$E[S_i] \;=\; (i - T_d + 1)\frac{1}{1-p} + T_d - 1.$$

Otherwise, if $i < T_d$, *then* $S_i = i$.

LEMMA 2.

$$Pr(R_i = t) = \begin{cases} (1-p)p^{n_{i,t}} & \text{if } (t - S_i) \mod T_d = 0 \\ 0 & \text{otherwise} \end{cases}$$

where $n_{i,t} = \lfloor (t - S_i)/T_d \rfloor$ *is the number of times packet* i *was lost when* $R_i = t$.

LEMMA 3.

$$Pr(R_i \leq t) = 1 - p^{n_{i,t}+1}.$$

Let D_v be the decoding time of a chunk v, and $P(v)$ be the set of chunks v depends on. Then, we have the theorem below.

THEOREM 1.

$$Pr(D_v = t) = \sum_{j \in \mathcal{P}(v)} \frac{Pr(R_j = t)}{Pr(R_j < t)} \prod_{k \in \mathcal{P}(v)} Pr(R_k < t) \tag{1}$$

The expected decoding time of a chunk v is thus

$$E[D_v] = \sum_{j=t}^{\infty} j Pr(D_v = j), \tag{2}$$

This model we developed previously allows us to estimate when a chunk can be decoded, considering dependencies and network characteristics. We can use this estimation to help us decide the sending order of the chunks. We proposed the following greedy heuristic in our previous work as well [Cheng, Ooi et al. 2007].

Suppose a chunk i has an importance w_i, as calculated from the previous section. We consider the chunks that have not been sent. For each chunk i, if all chunks that i depends on has either been pack or sent, we decide whether to send i in the current packet, or in the next packet. We compute a metric called *penalty* δ_i, given by

$$\delta_i = w_i(E[D_i^{next}] - E[D_i^{curr}]), \tag{3}$$

where D_i^{curr} and D_i^{next} are the decoding time of i if i is packed in the current packet and next packet respectively. Minimizing the penalty maximizes the difference in decoded plant quality. The greedy heuristic therefore simply packs the chunk with highest penalty at each step. We shall see in Section 4.3.2 the advantage of the greedy strategy when an important chunk is lost.

4. EXPERIMENTS

In order to validate our multi-resolution coding scheme, we have at first evaluated the resulting compressed representation, then tested the multi-resolution interdependent organization of the binary data over a lossy network.

Compression and streaming have been applied to three plants. We have used two digitized plant models: a 20 year old *Walnut* tree [Sinoquet et al. 1997] and an apple tree [Costes et al. 2003]. The walnut tree is 7.5m high and 5.8m wide. It took two weeks to digitize using a Polhemus 3Space Fastrack electromagnetic device. We pre-processed it by fitting Bézier curves to a series of digitized points representing branches. Our representation is thus composed of approximatively 1900 branches

with 6900 control points for the branches and 5800 control points for the radii. The apple tree is 6 year old, 2.8m high and 2m wide and is made of 430 branches, 1350 control points for the branches and 1100 for the radii.

To extend our experimental range of models, we have also generated some examples using L-systems. For example, we used here a fir-like tree composed of 6945 branches, 208,354 control points for the branches and 13900 control points for the radii. Of course, if used in an application, L-systems models would have been more efficiently transmitted by sending their generative rules and parameters. It supposes however that the client has enought computational power to also simulate plant growth from rules, which may not be the case for a light client (like PDAs). And determining generative process of a given tree is not always possible, in particular for measured tree or with construction process parametrized by direct user interaction.

4.1 Progressive decoding and grouping policies

As explained in Sections 3.2.2 and in 3.3.2, different setups and parameters for the grouping policy and for the quality metric can be defined to optimize different criteria. Experiments with the Walnut made us choose a *best compromise* grouping, but one should keep in mind that some criteria may be subjective and depend on the plants on which they are applied.

For the grouping policy of the *Walnut*, we have determined three setups, for the three following criteria:

—**Best compression** is achieved by reducing the degree of branches and radii to degree 2 and not doing any further grouping. In this case the compression ratio is 3.293).

—**Minimal quantization error** is obtained by degree-based grouping followed by heavy hierarchical clustering.

—**Best visual impact of the progressive decoding**, despite being a subjective criterion, may be obtained by reducing the degree to 1 and 2, and then using *scale-based grouping* for the branches (not the radii). Degree reduction leads to light models, therefore, the receiver has more information to decode upon receiving the same number of bits. Additionally, scale-based grouping allows bigger branches to *contribute a better approximation* of the intermediate tree earlier.

As the latest setup ensures both a good compression ratio and an acceptable quantization error, we define it as our *best compromise* and use it for all the experiment results we provide (e.g. in Section 4.2).

Regarding the quality metric, we have chosen $k_0 \gg k_2$ and $k_0 \gg k_1$, so that all models are sent before the instances and detail vectors. On Figure 10(d), degree-based grouping is applied. Models have an arbitrary number of control points and are therefore larger. A delay is noticeable: at 5% of the data, no branch instances have been decoded yet. In this case, k_0 may be lowered if visualizing models with very low percentage of the data is likely. When degree reduction is applied, the models are much lighter, and sending models first does not delay much the sending of branches (rows (a), (b) and (c)).

Tree name	Size (Bytes) and compression ratio		
	Basic	**Basic + bzip2**	**Our method**
Walnut	143608	84519 (1.70)	44098 (3.26)
Apple tree	28404	16026 (1.77)	9766 (2.91)
L-System (fir)	2666968	2358353 (1.13)	269108 (9.91)

Table III. Comparison of coding performance of three methods: basic binary coding, basic coding compressed with bzip2 and our progressive coding (using the "best compromise" setup; not the "best compression" one c.f. Section 4.1). Size is given in bytes and compression ratio is given w.r.t. the *basic* serialization size.

Moreover, for the ratio between k_1 and k_2, experiments have lead us to define two main strategies:

—The $k_1 \gg k_2$ strategy ensures that all instances are decoded before any details (priority to the *number* of branches).

—The "*cost per bits*" strategy creates a relationship between the size of the binary chunks and their importance. The (k_1, k_2) knobs are chosen inversely proportional to the average size of a instance chunk and a details chunk respectively (c.f. Table II).

Figure 10 shows progressive renderings of the *Walnut* for the main strategies we have defined. In row (c), we notice a significant change in shape between 40% and 85% of the data. This is due to the change of shape of a major, long branch bearing many children branches. Although the grouping strategy gives a good result in term of compression for the full model, the visual quality of intermediate tree is more satisfying in rows (a) and (b) with the *best compromise* grouping.

4.2 Compression of Plants

In order to appreciate the efficiency of our compressed model we have chosen to compare it with a well-known compression method (bzip2). Results are shown in Table III. Second column contains the size of a basic serialization of geometry and topology of the Bézier n-tree (with floats and integers coded on 32 bits). Third column shows the performance after compression with the well-known bzip2, one of the most efficient generic compression tools. Fourth column shows results for our method if binary chunks are concatenated. Even if simple, this concatenation keeps an important property of our model: it is progressive. Naturally this is appropriate for either file storage (with progressive loading) or network transmission (with progressive rendering on client). Moreover, we could save a little more by decreasing the pointer overhead: if binary chunks are completely ordered, then some IDs (e.g. instance and detail IDs) can be safely removed.

Results of Table III show that, for the *best compromise* grouping policy and for six bits per differential coordinate quantization (i.e. $c = 6$) on the *Walnut*, bzip2 compression applied to the basic coding has a 1.70 compression ratio, whereas our coding method brings it up to 3.26.

4.3 Transmission of a Set of Trees

As the main goal of our coding scheme is the progressive transmission of large natural scenes, we evaluated our transmission schemes over a lossy network to see

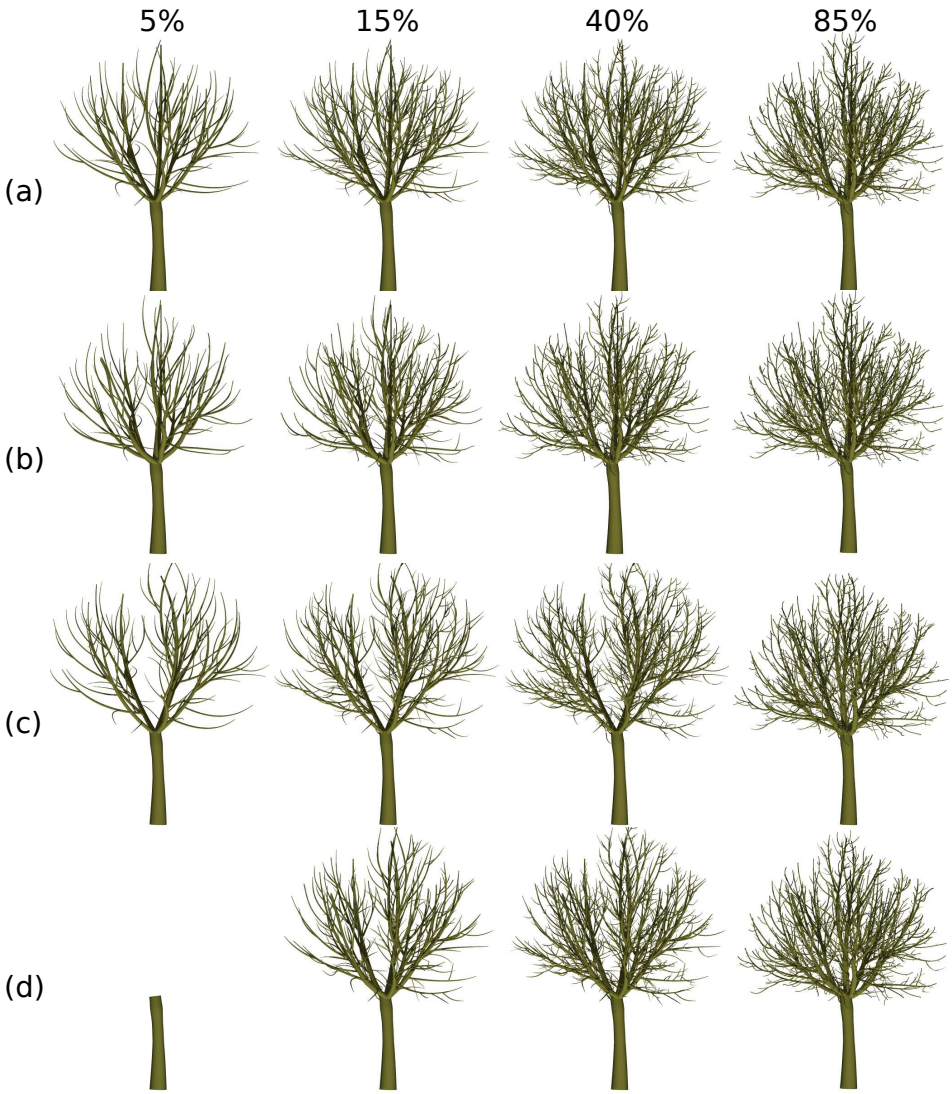

Fig. 10. Rendering after progressive decoding of the *Walnut*, for different setups: (a) *best compromise* with $k_1 \gg k_2$ strategy (b) *best compromise* with "*cost per bits*" strategy (c) best compression with $k_1 \gg k_2$ strategy (d) minimization of quantization error with $k_1 \gg k_2$ strategy

how our interdependent binary chunks can be efficiently packetized, transmitted over a lossy network, and progressively decoded.

4.3.1 *Experimental Streaming Setup.* We ran our experiments using a client-server streaming setup. On the server-side, the 3D scene data is loaded as a DAG of binary chunks (Figure 2). This DAG structure is packetized by the scheduler

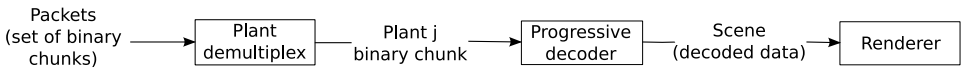

Fig. 11. The decoding process.

that implements a given strategy (c.f. Section 3.3.3) and transmitted. Lost packets are retransmitted; the retransmission order differs depending on the strategy. On client-side (Figure 11), binary chunks are demultiplexed between objects and a progressive decoder associated with the plant decodes them as soon as possible for rendering (as in Figure 10).

We used the unreliable transport protocol DCCP (c.f. [Kohler et al. 2006]) with retransmission. With an unreliable protocol, packets comming after a loss can still be processed before repairing the loss, unlike TCP which buffers packets in order to maintain the original ordering and thus waste potential bandwitdh. We have chosen DCCP over UDP because it is the recommended way of tranporting unreliable datagrams while being friendly with concurrent TCP streams. For reproducibility, we first captured a packet trace of random packet data over given network conditions, and used the trace to simulate transmissions of plants over realistic network conditions and replayed it locally while decoding progressively the packetized plants.

Using of DCCP required special care for our experiments. First, because of security restrictions on firewalls, DCCP ports needs to be opened. Moreover because DCCP uses TCP's port's system (client receives data on a random port), configuring port traversal via firewalls is needed. The second problem came from the backbone itself: experiments carried out between Toulouse and Singapore showed that some backbone routers do not route DCCP traffic. The solution to the last problem was to implement an application-level UDP tunnel that carries out DCCP traffic while keeping loss rate's influence on the DCCP stack.

Two main congestion control mechanisms have been implemented in DCCP (CCID2 and CCID3 in DCCP's RFC [Kohler et al. 2006]). CCID2 implements TCP-like congestion control whereas CCID3 uses TFRC (TCP-friendly Rate Control). We have chosen to present here experiments with CCID2, as we found its implementation in recent Linux kernels more robust.

Experiments with DCCP have been carried out on both WAN (with the tunnel) and a LAN simulating a WAN (by adjusting packet loss and delay with traffic control tools of the Linux kernel). For conciseness, we shall only present here results on a real WAN.

4.3.2 *Transmission Results.* As an example of our experiments, we give results for the transmission of a set of four *walnut* trees (considered independent from each other) over the Internet, between Toulouse and Singapore. The dynamic importance proposed at the end of Section 3.3.2 is used. The importance is *scaled* by the distance to the view point so that the binary chunks are interleaved with respect to their closeness to the viewer and their static quality. We used DCCP with retransmission (with CCID2) in our UDP tunnel (c.f. Section 4.3.1). For this network capture, we have measured a packet loss weaving between 10% and 15%.

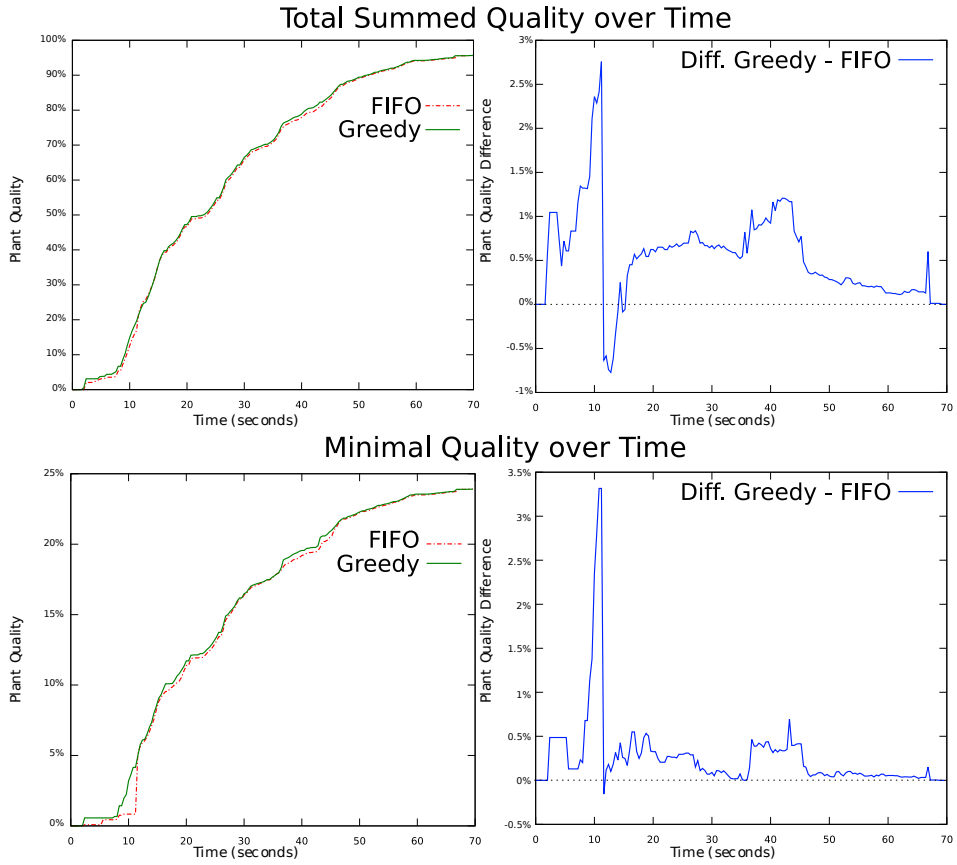

Fig. 12. Comparison of the efficiency of packetization strategies *Greedy* and *FIFO*, during the progressive transmission of four walnuts between Toulouse and Singapore. The minimal quality correspond to the tree having the minimal quality among the four. The right figure plot the difference in quality between the two strategies.

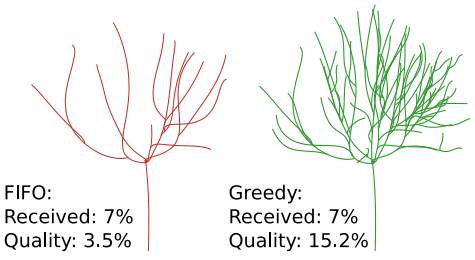

Fig. 13. Example showing the structure of the same tree of the scene better packetized by the *Greedy* strategy.

Two packetization strategies have been tested: *FIFO* strategy, which takes into account both dependencies and importance ordering between binary chunks, and the proposed *Greedy* strategy (see Section 3.3.3).

In Figure 12, we show the evolution of the quality of the set of trees (in the first row, the sum of the tree qualities; in the second row, the quality of the tree of minimal quality) over time for *FIFO* and *Greedy* strategies (the right column shows the difference between these two curves). Both experiments use the same set of packet trace. So, at a given time, both have received exactly the same amount of data. On the arrival of a packet, we reconstruct the trees with the available binary chunks and compute the quality. Therefore, the plots tend to confirm that the proposed *Greedy* strategy of the binary chunks improves the amount of decodable data over transmission time. Figure 13 shows the reconstruction of one of the trees after receiving 7% of its data during both transmissions. We can observe that most of the data received in the *FIFO* case is unusable due to the lack of one or more binary chunks on which many others depend. The aim of the *Greedy* packetization strategy is rightly to prevent those "accidents" to happen.

5. CONCLUSION AND PERSPECTIVES

We have proposed an original progressive representation of branching systems adapted to the streaming of 3D scenes. This representation allows efficient compression of the plant geometry represented by generalized cylinders. Our method outputs a set of interdependent binary pieces of data well suited for packetization and progressive transmission over lossy networks with the help of a quality metric.

There are several directions we can take to continue this research.

For the progressive representation of plants, we may design more accurate methods for grouping Bézier curves based on their geometry (and not on their control points). For example we can analyse more precisely the shape of the curves, using PCA (Principal Component Analysis), Curvature Scale Space (c.f. [Mokhtarian et al. 1996]) or minimal energy surfaces (c.f. [Osserman 1986]). Moreover, it would be interesting to find out the effectiveness of sharing branch models between different plants of the scene, i.e. *Forest-based* progressive compression. The challenge is to have more accurate models while keeping a good ratio between the numbers of models and instances.

For progressive transmission of plants, the quality metric could be made more *dynamic* by considering the viewpoint of the navigating user more accurately. For example, at the scene level, scene data close to the central region of the view frustum should have a higher importance and therefore be streamed first. The current quality metric also depends on the tunning of parameters k_0, k_1 and k_2 (Section 3.3.2). A user survey may be useful to evaluate the subjective impact of the quality metric, especially in the presence of undesired effects such as *popping*, a common problem of most 3D multi-resolution models.

The efficiency of our progressive representation could be evaluated in other applications requiring progressive models for plants, for example 3D visualisation on mobile devices or plant modeling or animation software.

Finally, in order to have fully fledge trees, we also need to consider standardization and instantiation to represent leaves. Our tree representation can be naturally extended with some reference leaf symbols that could be instantiated similarly to branches. In other words, leaves can be considered as special branches where geometrical information is used to place leaf models on branches. Leaf models can

themselves be geometrically defined as Bézier patch. Similar scheme to decompose them as models and instances with difference vectors of control points can thus be produced.

6. ACKNOWLEDGMENTS

This work has been partly funded by the *NatSim* ANR project (French National Research Agency 05-MMSA-0004-01) and National University of Singapore Academic Research Fund R-252-000-306-112.

REFERENCES

Al-Regib, G. and Altunbasak, Y. 2003. 3TP: an application-layer protocol for streaming 3D graphics - experimental approach. In *IEEE Int. Conf. on Multimedia and Expo*. Vol. 1. 421–424.

Al-Regib, G., Altunbasak, Y., and Rossignac, J. 2002. An unequal error protection method for progressively compressed 3D models. In *Int. Conf. On Acoustics, Speech, and Signal Processing*. Vol. 2. 2041–2044.

Behrendt, S., Colditz, C., Franzke, O., Kopf, J., and Deussen, O. 2005. Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum 24,* 3 (September), 507–516.

Bloomenthal, J. 1985. Modeling the mighty maple. *ACM Computer Graphics (SIGGRAPH'85) 19,* 3 (July), 305–311.

Bogacki, P., Weinstein, S. E., and Xu, Y. 1995. Degree reduction of Bézier curves by uniform approximation with endpoint interpolation. *Computer-Aided Design 27,* 9, 651–661.

Boudon, F., Meyer, A., and Godin, C. 2006. Survey on Computer Representations of Trees for Realistic and Efficient Rendering. Tech. rep., LIRIS UMR 5205 CNRS.

Chen, Z., Barnes, F., and Bodenheimer, B. 2005. Hybrid and forward error correction transmission techniques for unreliable transport of 3D geometry. *Multimedia Systems 10,* 3, 230–244.

Cheng, I., Ying, L., and Basu, A. 2007. Packet-Loss Modeling for Perceptually Optimized 3D Transmission. *Advances in Multimedia 2007*, Article ID 95218, 10 pages.

Cheng, W. and Ooi, W. T. 2008. Receiver-driven View Dependent Streaming of Progressive Mesh. In *Proceedings of the 18th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*.

Cheng, W., Ooi, W. T., Mondet, S., Grigoras, R., and Morin, G. 2007. An analytical model for progressive mesh streaming. In *The 15th international Conference on Multimedia, ACM, Augsburg, Germany, 24/09/07-29/09/07*. ACM Press, 737–746.

Costes, E., Sinoquet, H., Kelner, J.-J., and Godin, C. 2003. Exploring within-tree architectural development of two apple treecultivars over 6 years. *Annals of Botany 91*, 91–104.

Decaudin, P. and Neyret, F. 2004. Rendering Forest Scenes in Real-Time. In *Proceedings of the 15th Eurographics Symposium on Rendering*. 93–102.

Deussen, O., Colditz, C., Stamminger, M., and Drettakis, G. 2002. Interactive visualization of complex plant ecosystems. In *Proceedings of IEEE Visualization*.

Deussen, O. and Lintermann, B. 2005. *Digital Design of Nature: Computer Generated Plants and Organics*. Springer-Verlag.

Farin, G. 2002. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann Publishers Inc.

Galbraith, C., MacMurchy, P., and Wyvill, B. 2004. BlobTree Trees. In *Proceedings of Computer Graphics International*. 78–85.

Harris, A. and Kravets, R. 2002. The design of a transport protocol for on-demand graphical rendering. In *NOSSDAV '02: Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. 43–49.

Hoppe, H. 1996. Progressive meshes. In *SIGGRAPH'96: Proceedings of the 23rd Annual Conference on Computer graphics and interactive techniques*. 99–108.

Johnson, S. C. 1967. Hierarchical Clustering Schemes. *Psychometrika 2*, 241–254.

Kohler, E., Handley, M., and Floyd, S. 2006. Datagram Congestion Control Protocol.

Li, H., Li, M. Y. B., and Prabhakaran, B. 2006. Middleware for streaming 3D progressive meshes over lossy networks. *ACM Trans. Multimedia Comput. Commun. Appl. 2,* 4, 282–317.

Meng, F. and Zha, H. 2003. Streaming Transmission of Point-Sampled Geometry Based on View-Dependent Level-of-Detail. *International Conference on 3D Digital Imaging and Modeling*.

Meyer, A., Neyret, F., and Poulin, P. 2001. Interactive Rendering of Trees with Shading and Shadows. In *Proceedings of the Eurographics Workshop on Rendering Techniques*. 183–196.

Mokhtarian, F., Abbasi, S., and Kittler, J. 1996. Robust and efficient shape indexing through curvature scale space. In *Proceedings of British Machine Vision Conference*. 53–62.

Mondet, S., Cheng, W., Morin, G., Grigoras, R., and Ooi, W. T. 2008. Streaming of Plants in Distributed Virtual Environments. In *MULTIMEDIA '08: Proceeding of the 16th ACM international Conference on Multimedia*. ACM, 1–10. Best Paper Award.

Neubert, B., Franken, T., and Deussen, O. 2007. Approximate image-based tree-modeling using particle flows. *ACM Transactions on Graphics 26,* 3 (July), 88.

Osserman, R. 1986. *A Survey of Minimal Surfaces*. Dover Publications, New York.

Park, S.-B., Kim, C.-S., and Lee, S. U. 2006. Error Resilient 3-D Mesh Compression. *IEEE Transactions on Multimedia 8*, 885–895.

Prusinkiewicz, P. 1986. Graphical applications of L-systems. In *Vision Interface*. 247–253.

Prusinkiewicz, P. and Lindenmayer, A. 1990. *The algorithmic beauty of plants*. Springer Verlag.

Prusinkiewicz, P., Mündermann, L., Karwowski, R., and Lane, B. 2001. The use of positional information in the modeling of plants. *ACM Computer Graphics (SIGGRAPH'01) 22,* 4, 289–300.

Ramanathan, P., Kalman, M., and Girod, B. 2003. Rate-distortion optimized streaming of compressed light fields. In *IEEE International Conference on Image Processing*. 277–280.

Remolar, I., Chover, M., Belmonte, O., Ribelles, J., and Rebollo, C. 2002. Geometric Simplification of Foliage. In *Eurographics'02 Short Presentations*. 397–404.

Rusinkiewicz, S. and Levoy, M. 2001. Streaming QSplat: a viewer for networked visualization of large, dense models. In *Symposium on Interactive 3D Graphics*. 63–68.

Sinoquet, H., Rivet, P., and Godin, C. 1997. Assessment of the three-dimensional architecture of walnut trees using digitising. *Silva Fennica 31,* 3, 265–273.

Tari, B., Yemez, Y., Ozkasap, O., and Civanlar, R. 2005. Progressive View-Dependent Transmission of 3D Models over Lossy Netowrk. In *Proceedings of the 13th European Signal Proceesing Conference*.

Tian, D. and AlRegib, G. 2004. FQM: a fast quality measure for efficient transmission of textured 3D models. In *MULTIMEDIA '04: Proceedings of the 12th Annual ACM International Conference on Multimedia*.

Weber, J. and Penn, J. 1995. Creation and Rendering of Realistic Trees. *ACM Computer Graphics (SIGGRAPH'95) 29,* 3 (August), 119–128.

Yan, Z., Kumar, S., and Kuo, J. 2001. Error resilient coding of 3D graphic models via adaptive mesh segmentation. *IEEE Trans. Circuits Syst. Video Technol 11*, 860–873.

Yang, S., Lee, C.-H., and Kuo, C.-C. J. 2004. Optimized mesh and texture multiplexing for progressive textured model transmission. In *MULTIMEDIA '04: Proceedings of the 12th Annual ACM International Conference on Multimedia*.

Zhang, X., Blaise, F., and Jaeger, M. 2006. Multiresolution plant models with complex organs. In *VRCIA'06: Proceedings of the 2006 ACM international Conference on Virtual Reality Continuum and Its Applications*. 331–334.

Received Month Year; revised Month Year; accepted Month Year